ChessBrain: a Linux-Based Distributed Computing Experiment

Carlos Justiniano
cjus@chessbrain.net, cjus34@yahoo.com

Published in the September 2003 issue of the Linux Journal
http://www.linuxjournal.com/article/6929

On May 27, 2003, 646 machines worked together to play a single game of chess. This was the first time such a feat had been accomplished, and it was made possible by the power of Linux, open-source software and hundreds of contributors from over 37 different countries.

ChessBrain (chessbrain.net) is a distributed computation project that uses the idle processing power of distributed machines to solve computationally intensive problems. ChessBrain is a system focused on playing chess, but the underlying system can be adapted for other games as well as for non-game-related applications.

Imagine playing a game against an opponent, except every time he moves, you grab the phone and start calling friends for help. You call Sue, describe the current position and ask her to call you back when she has an answer. Then you call Ryan to ask whether you should worry about a pending attack; again, you ask for a call back when he has an answer. After calling 20 other friends, you sit back and wait for replies. This is similar to how ChessBrain plays chess.

ChessBrain consists of a Linux-based server application, the SuperNode, and client software known as PeerNodes. The SuperNode connects to an on-line game server, which allows visiting members to play against one another, challenge ChessBrain to a game or watch ChessBrain play against its current opponent. While ChessBrain plays, it examines positions, dispatches hundreds of potential moves to remote PeerNodes for analysis, collects feedback from the PeerNodes, processes that information and makes its best move. ChessBrain exists as an ever-changing pool of networked machines. Philosophically and scientifically, it's a beautiful thing.

I started ChessBrain as a distributed computing experiment in the summer of 2001. By the end of that year, I had a working prototype and needed a place to host the server. My longtime friend, Walter Howard, the webmaster of HackerWhacker (hackerwhacker.com), offered to host the server on his personal T1 line.

Figure 1. The First ChessBrain Servers

On June 9, 2002, ChessBrain appeared on Slashdot, and the positive exposure resulted in hundreds of new PeerNode operators. Gavin Roy, one of the new members, owns the bteg network (http://www.bteg.net/) and offered to host a SuperNode server free of charge. On June 27, I met Gavin for dinner and handed this near stranger a SuperNode server on a Pentium III machine. ChessBrain gained another server, I gained another friend, and Gavin has become an important supporter of the ChessBrain Project. I transitioned the SuperNode over to Gavin's site, and Walt continued to host the original SuperNode as a secondary backup and experimental server.

During the months that followed, we gained an amazing amount of exposure. Few seemed to mind that ChessBrain couldn't actually play chess. The first eight months of 2002 were spent working on the SuperNode server and porting the PeerNode client to Microsoft Windows and Apple's Mac OS X.

Once the server and clients worked well, the focus was on getting ChessBrain actually to play. The wbec-ridderkerk (http://www.wbec-ridderkerk.nl/) site in the Netherlands lists nearly 200 freely available chess-playing programs. I reviewed a few, looking for one with relatively clean code

and the ability to compile under several operating systems. I found an ideal program in Beowulf, written by Colin Frayn, who was then a PhD candidate at Cambridge University in England. We exchanged several e-mails and Colin joined the project. We collaborated entirely on-line using e-mail and instant messaging (IM) and began making necessary modifications. Colin adapted his chess program for distributed computing, and I modified the SuperNode and PeerNode clients to use his engine. The time difference between London and Los Angeles proved ideal. I would IM Colin at my 3AM and again during the day. By my late afternoon, Colin would head for bed, and I would work through his night. Before crashing, I would leave feedback for Colin. This round-the-clock cycle continued for months.
Colin adapted his original Beowulf chess engine to become two chess-playing components, BeoServer and BeoClient. He developed the pair to support distributed chess play within the ChessBrain framework. On December 22, 2002, ChessBrain played its first game of distributed chess. By January 2003, the ChessBrain community had provided 62 machines and was testing regular builds.

Overview

The SuperNode and PeerNode are multithreaded applications written in C++ and compiled using GCC under Red Hat Linux 7.1, 7.2 and 8.0. The primary SuperNode server runs under Slackware 8.0 at bteg network's colocation site in Northern California (Figure 2).

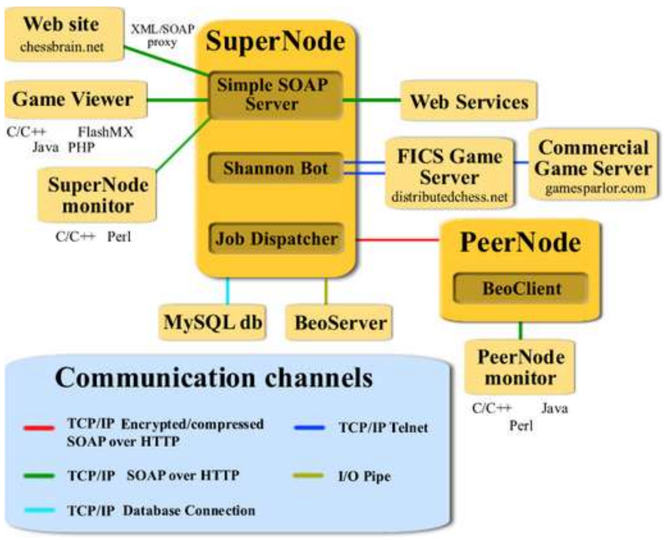

Figure 2. The ChessBrain System Architecture

Because the applications are heavily multithreaded, I spent a fair amount of time resolving threading issues. I used GDB, DDD and custom logs to tackle debugging. Early in the development process, Perl scripts proved especially effective in helping test new functionality and stress test the software. I have 12 machines at home; these, plus an army of Perl scripts pounding on a local server, proved to be formidable testing tools.

XML, SOAP and Web Services

Early in the project I realized the SuperNode server would need to communicate with other servers. During that time XML offered a viable approach, and later XMLRPC (http://www.xmlrpc.org/) brought additional advantages. The Simple Object Access Protocol (SOAP) continued evolving to meet the needs of servers that speak to other servers. Encouraged by promises of improved interoperability, I adopted SOAP as the preferred method of communication for the SuperNode server and PeerNode client.

From the outside, the SuperNode acts like a Web server with SOAP-based interfaces. Although the SuperNode server handles HTTP GET and POST, the POST message is used most often. The SuperNode parses HTTPs and XML-based SOAP requests, processes those requests and returns HTTP packages with embedded SOAP payloads.

The SuperNode and PeerNode parse SOAP requests and route commands to an internal command dispatcher, which ensures that the correct command handlers process the requests. In the SuperNode, the most common requests come from PeerNode clients; a PeerNode must connect to request a job unit. A job unit is an XML block containing a game position and instructions on how to analyze the position. A PeerNode contains a complete chess engine component, compiled and linked as a static library. When the PeerNode receives a job unit, it processes the SOAP response, extracts the job-specific information and passes instructions to its internal chess component for analysis.

The SuperNode server then passes the current game position to the external BeoServer process. Interprocess communication between the SuperNode and BeoServer is accomplished using two pipes. In the near feature, we expect to move BeoServer to its own box and shift to UDP over 1000Base-T Ethernet.

Security

Secure and tamper-free communication is a necessity for ChessBrain. An invalid result created by a malicious user could render the play ineffective and ultimately embarrassing. Sensitive communication is protected using the Advanced Encryption Standard, AES Rijndael (pronounced Rhine-doll). AES is a variable block symmetric encryption algorithm developed by Belgian cryptographers Joan Daemen and Vincent Rijmen as a replacement for the aging DES and Triple DES standards.

Before exploring Rijndael, the Blowfish symmetric cipher was used until the PeerNode client was ported to Mac OS X and problems surfaced involving endian issues with the implementation of Blowfish being used. AES is an endian-neutral algorithm and proved ideal for our situation.

The original design of the PeerNode involved having the client and its chess engine as two separate processes. The PeerNode started the chess engine process and redirected the standard I/O to establish a loose binding. Initially, we avoided directly linking chess code with the PeerNode client so the chess code could be replaced quickly and easily in future iterations of the software. We later moved to a static linking approach to deal with potential security issues. The problem was that it's entirely possible to write a chess engine proxy that sits between the PeerNode and the actual chess engine program. This would offer an easy way to alter results before sending them to the SuperNode server. We decided to link the engine component statically because of two key advantages, tighter security and function-based rather than I/O-based messaging.

The surge of interest from Slashdot soon made it necessary to reduce ChessBrain's bandwidth requirements. To this end, the use of SOAP offered many advantages, but its size left much to be desired. The Zlib data compression library (http://www.zlib.org/) is now used prior to encryption to reduce the size of SOAP-based messaging. Naturally, adding compression and encryption reduces the potential for interoperability; however, the XML encryption specification (www.w3.org/TR/xmlenc-core) offers an alternative approach.

Bots, Presence and Autonomous Play

The SuperNode server has a Bot called Shannon (implemented as a thread) that connects to on-line game servers and maintains a presence. Members of the game server type commands to challenge ChessBrain or to watch the current game being played. It has been fun programming Shannon, which now understands a variety of commands. There is a great deal of potential in on-line bots that can be instructed to perform actions on behalf of their hosts.

During the development of ChessBrain, I downloaded the source code to a Free Internet Chess Server (FICS) and compiled it on an old Pentium 200 MMX Toshiba laptop running Linux. FICS is written in C, and it compiled using GCC without incident. The game server allows users to telnet to port 5000 and sign in with a user name and password. After a few months the traffic increased, and we moved the FICS server to another ChessBrain machine at our secondary domain at distributedchess.net. Users now have several options for watching ChessBrain play on-line. They can telnet directly to the game server where ChessBrain is playing or use one of our viewer programs.

After ChessBrain could play on an on-line game server, I wrote a Java game viewer to allow people to watch live games. As an alternative to the Java viewer, I also wrote viewers based in PHP and Macromedia Flash (www.chessbrain.net/viewers.html). ChessBrain contributor Anthony Bravo wrote a Java-based network viewer to show the active PeerNodes throughout the world. Users can click on nodes to see how many machines are active in a given country. All of the viewers on the ChessBrain site use SOAP to communicate with the SuperNode.

As a security precaution, browser plugins such as Java and Macromedia's Flash ActionScript don't allow the program to connect to a server other than the one from which the applet was downloaded. To work around this issue, I wrote a simple XML proxy script that accepts an HTTP GET request on one server and connects to the SuperNode server on behalf of the client. For example, if you wanted to query the SuperNode server for the current game position, you could enter the following URL into your browser:

```
http://www.chessbrain.net/xmlproxy.php?command=CBSG
etPos
```

The server would respond with a SOAP package like the one in Figure 4. On Mozilla you can view the page source to see the actual SOAP document.

Listing 1. A ChessBrain XML Response Package

```
<?xml version="1.0" ?>
<env:Envelope xmlns:env=
  http://www.w3.org/2001/12/soap-envelope
  xmlns:enc=
    "http://www.w3.org/2001/12/soap-encoding">
  <env:Body>
    <cbs:CBSGetPosResponse>
      <return>
        rn1qk2r/p2p1ppp/bb2pn2/1p6/1P6/
        P2Q1NP1/1BP1PP1P/RN2KB1R b KQkq -
      </return>
    </cbs:CBSGetPosResponse>
  </env:Body>
</env:Envelope>
```

Monitoring the SuperNode

Monitoring a server's health is an important part of system administration. Fortunately for developers, Linux offers many ways to tackle server monitoring. The Linux /proc virtual filesystem contains a goldmine of valuable system data, offering developers an easy way to profile and monitor system behavior. /proc/net/dev offers device data such as the number of bytes and packets sent and received on a network interface, and /proc/meminfo offers loads of memory statistics. If data mining the /proc isn't your thing, sysinfo() offers a quick and easy way to fill a structure with system statistics, such as system load, freeram and the total number of processes.

The SuperNode server offers a SOAP request that returns system information similar to what is shown in Listing 2.

ChessBrain member Greg Davis wrote the first SuperNode monitor in Perl, which issues the SOAP request and displays a screen similar to the top command.

Listing 2. An XML Server Status Message

```
<?xml version="1.0" ?>
<env:Envelope>
  <env:Body>
    <cbs:CBSSysInfoResponse
      >
    <Uptime days="1" hours="14"
            minutes="43" seconds="18" />
    <System proccnt="546" totmem="250.13"
            freemem="4.38"
            memu="98"
            cpustates="3627078,0,2151891,8160852"
            loadavg="0.50,0.30,0.33" />
      <Recv Bytes="2301480887.000000"
            Packets="16652816.000000"
            Errors="0.000000"
            Drop="0.000000" />
      <Send Bytes="2443488824.000000"
            Packets="12142245.000000"
            Errors="0.000000"
            Drop="0.000000" />
     </cbs:CBSSysInfoResponse>
   </env:Body>
</env:Envelope>
```

PeerNode Monitoring

Because many members of the ChessBrain community run PeerNode clients on several machines, they wanted a convenient way to monitor the status of a group of machines. The PeerNode client was modified to post SOAP requests to port 3434 so PeerNode monitor applications could listen on that port and display real-time status information. I wrote the first PeerNode monitor application, and other members submitted their own. Greg Davis and Oliver Otte both submitted Perl-based PeerNode monitors. The most popular PeerNode monitor, CBMoc, was written in Java by Kris Drent.

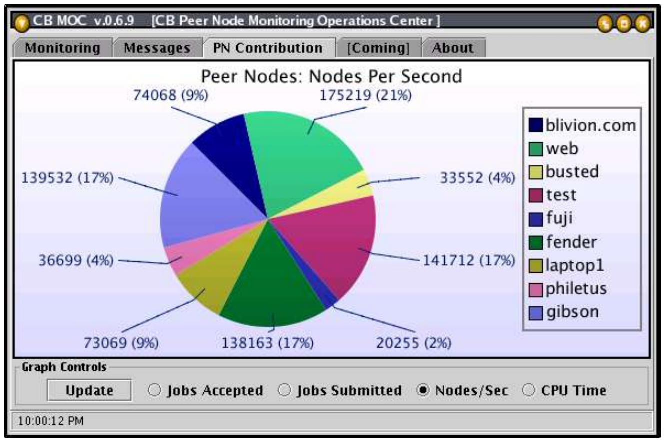

Figure 3. Java-Based CBMoc

Cross-Platform Graphics and Data Visualization

ChessBrain collects a great deal of data, and we're currently exploring data visualization as a way of tracking and investigating system behavior. We're adding 3-D-based game navigation tools to allow us to track network play visually. Sven Herrmann, our resident 3-D expert, has created an OpenGL-based renderer that we use in our SuperNode monitor application. We'll also use the new renderer in our next-generation screensaver program and game viewers.

Figure 4. Real-Time OpenGL Rendering

Conclusion

Today, ChessBrain is a working prototype that plays chess using hundreds of machines running Linux, FreeBSD, Mac OS X and Microsoft Windows. It plays at international master strength, and we see many ways to continue making improvements.

ChessBrain exemplifies the use of open-source tools to solve complex problems. We're preparing ChessBrain itself for release as an open-source project, with the hope that others will join and expand the effort. The wonderful thing about ChessBrain is that there are so many ways to contribute. Anyone with a PC and Internet connection can participate. Download a PeerNode client from the ChessBrain Web site, run the software on one or more computers and help increase the size of ChessBrain.

We're currently working toward an official world record for the largest number of machines used to play a single game. We have an established contact at the World Record office in London and contacts within a number of official chess federations. The ChessBrain Project is supported by a strong core team, including Peter Wilson, the former chairman of the World Chess Federation's (FIDE) Computer Chess and Internet Committee. The ChessBrain team currently is exploring potential venues for a public and Internet-based demonstration involving a new Guinness World Record in distributed computation and chess. Check the ChessBrain site for an official announcement.

Acknowledgements

Many thanks to Janus Daniels, Cedric Griss and other ChessBrain team members for their support in the preparation of this article. For contact information and a list of who's who on ChessBrain visit:
www.chessbrain.net/friends.html.